**Grimoire of the Destined (working title) v0.9**

**DESIGN DOCUMENT – ENGINE**

**~~The engine will be in OpenGL. The reason for this is that currently Microsoft isn’t doing as well as it should, so we should opt for open-source technologies rather than proprietary. It would be nice to learn about DirectX at a future date, but by sticking with something proprietary, you’re running the risk of learning a useless skill when that company goes under and the technology doesn’t become industry-standard.~~**

**Required Environment:**

* **FlashDevelop IDE using ActionScript 3**
* **Flixel**
* **32-bit Java**
* **Flash Debug Player (installable by toggling an option when installing the IDE)**

**To set up the project:**

**Open an ActionScript 3 project and right-click the project in the explorer to your right. Go to Properties under the Classpaths tab. Add a classpath by clicking the directory where the Flixel library files are. Once you apply or press OK, the library should be ready to use. On the top of each of your .as files, after the package declaration, include this import statement…**

**import org.flixel.\*;**

**This will allow you to use the entire Flixel library.**

**Input**

**Input should be keyboard only. I believe libraries for console controllers such as XBOX and PS3 are proprietary, and I don’t think most PC gamers play RPGs using joypads or joysticks.**

**By default, these should be the mapped keys:**

* **Arrow keys to move.**
* **Right shift to toggle run.**
* **X to confirm**
* **Z to cancel**
* **A for left trigger**
* **S for right trigger**
* **C for field menu**

**~~Entity base class~~**

**~~All drawable entities must be derived from a base class. The base class will have properties that all other entities have in common, such as colour, alpha value, xy-coordinates, and the like. All entities must have a single draw function, which will simply draw the image on-screen. Animations will be done in higher layers by setting which image should be drawn next based on numeric index.~~**

**Placeholder graphics and fonts**

**To test the graphics in our engine, we could start with coloured squares with different borders to discern that it works. For animated graphics, we could make a spritesheet of squares with named frames to show that that is the frame that is supposed to show.**

**States**

**A graphics engine must have states. Essentially, a state is like a scene that you draw on-screen at the window. You can have states on top of states, but only the state on top is being redrawn, and the bottom state is static.**

**States should be stored in an actual stack. Throughout the game, states will be pushed and popped frequently, so it should be fast, therefore we chose C++ for this implementation.**

**The game will have the following states…**

* **A menu state. This isn’t just for a main menu, but for a character menu, a shop menu, exit menu, or dialogues. Only panels and buttons will be drawn in this state. If there are a series of buttons, they could be conveniently put into a collection for easy reference. This state can be drawn on top of other states.**
* **A field state. This will be for the overworld, inside towns, as well as caverns and the like. Stationary objects, passable and impassable textures will be drawn, as well as animated entities such as fireplaces and people. Events will also be drawn in this state.**
  + **Each of these objects, textures, and entities should be put in their own collections for easy reference.**
* **A battle state. The fights are obviously going to be drawn in this state. Enemies will be drawn on the left side, and party members will be drawn on the right side, with a foreground to show the terrain where the battle is commenced. Events will be drawn in this state if applicable. Like field state, drawable entities should be stored in their own collections for easy reference.**
* **An interlude state. Kind of a specialized state. White text on black screen with some colour effects, mostly, to progress the story and add dramatic effect. Nothing else should be drawn in this state. The drawing of text may be timed and uninterruptible.**

**Transitions**

**A JRPG isn’t a JRPG without transitions. We definitely want the following.**

* **Fadein and fadeout. These are the easier transitions because it simply requires a gradual increase and decrease of alpha value. You want to maintain a constant value as the state changes to make the transitions symmetric to each other.**
* **Checker slide-in and slide-out. Basically, small square portions of pixels fadeout from left to right, then fadein from right to left, or similar but from the center outward. Might be complicated because it requires you to manipulate individual pixels rather than an entire image’s alpha value.**
* **Whiteout or Blackout. Optional, but easy to implement. Simply wash-out the entire screen in white or black, then fadeout in the case of whiteout.**

**~~Sound~~**

**~~Obviously, a sound manager should be implemented. There are two kinds: sounds and BGM, which go on separate track layers. Sounds can play in a timed queue and play only once. BGM can play once or loop depending on configuration. There should be an in-game volume control. Whether or not it’s possible to loop mp3 files, we should have support for both mp3 and wav files. Wav files are loopable, but very bulky. If there are loopable, small sound file formats that we can use, we can support it for the purposes of this engine.~~**

**~~Game Timer~~**

**~~As soon as the game runs, a game timer should also be running that will allow events to run based on a moving frame of reference. The game timer will be in milliseconds, but should also have functions to convert it to seconds, minutes, or hours. Game saves will keep track of playtime in seconds, and should have a limit to prevent overflow.~~**

**Framerate and Screen Resolution**

**Should be considered in terms of implementation for different screens. 60 is our target framerate, and we should have at least 3 standard resolutions (640x480, 800x600, 1024x768)**

**Encryption**

**~~There has to be a way to encrypt data and save files to a defined format without using easily known ways. Though encryption is kind of futile, we still want to encrypt it so that it’s not easily tamperable.~~**

**Save files are supported in Flixel, but as for data files, the most secure way of doing it is to put it in the code as a fixed collection, or we encrypt the files ourselves using a known encryption/compression scheme.**